

Stochastic Analysis of a Churn-Tolerant Structured Peer-to-Peer Scheme

Tim Jacobs *

Gopal Pandurangan †

Abstract

We present and analyze a simple and general scheme to build a churn (fault)-tolerant structured Peer-to-Peer (P2P) network. Our scheme shows how to “convert” a static network into a dynamic distributed hash table(DHT)-based P2P network such that all the good properties of the static network are guaranteed with high probability (w.h.p). Applying our scheme to a cube-connected cycles network, for example, yields a $O(\log N)$ degree connected network, in which *every* search succeeds in $O(\log N)$ hops w.h.p., using $O(\log N)$ messages, where N is the expected stable network size. Our scheme has a constant storage overhead (the number of nodes responsible for servicing a data item) and an $O(\log N)$ overhead (messages and time) per insertion and essentially no overhead for deletions. All these bounds are essentially optimal. While DHT schemes with similar guarantees are already known in the literature, this work is new in the following aspects: (1) It presents a rigorous mathematical analysis of the scheme under a general stochastic model of churn and shows the above guarantees; (2) The theoretical analysis is complemented by a simulation-based analysis that validates the asymptotic bounds even in moderately sized networks and also studies performance under changing stable network size; (3) The presented scheme seems especially suitable for maintaining dynamic structures under churn efficiently. In particular, we show that a spanning tree of low diameter can be efficiently maintained in constant time and logarithmic number of messages per insertion or deletion w.h.p.

Keywords: P2P Network, DHT Scheme, Churn, Dynamic Spanning Tree, Stochastic Analysis.

1 Introduction

Peer-to-Peer (P2P) networks are highly dynamic: peers enter and leave the network and connections may be added or deleted at any time and thus the topology changes very dynamically. The independent arrival and departure by a large number of peers creates a collective effort that is called as *churn*. Measurement studies of real-world P2P networks [25, 26, 29] show that the churn rate is quite high: nearly 50% of peers in real-world networks can be replaced within an hour. However, despite a large churn rate, these studies show that the total number of peers in the network is relatively *stable*. The study by Stutzbach and Rejaie [29] also indicate that P2P networks exhibit a high degree of variance in terms of the session time (the amount of time spent by a node in the network in a session). They show that the distribution of session times appear to follow a Weibull or lognormal distribution.

P2P systems must have efficient and reliable routing in the presence of a dynamically changing network. The P2P overlay must exhibit good topological properties (e.g., connectivity, low diameter, low degree, etc.) even if the composition of the underlying physical network exhibits significant change. Because the system dynamics of these networks are also highly asymmetric with only a small number of peers persistent over

*Department of Computer Science, Purdue University, West Lafayette, IN, 47907, USA.

†Division of Mathematical Sciences, Nanyang Technological University, Singapore 637371 and Department of Computer Science, Brown University, Providence, RI 02912, USA. E-mail: gopalpandurangan@gmail.com. Supported in part by NSF grant CCF-0830476.

significant time periods, providing churn-tolerance in the presence of mostly short-lived peers is essential [29].

A Distributed Hash Table (DHT) scheme (e.g., [21, 28, 15]) creates a fully decentralized index that maps data items to peers and allows a peer to search for a data item very efficiently (typically logarithmically in the size of the network) without flooding. Such systems have been called *structured* P2P networks, unlike Gnutella, for example, which is *unstructured*. In unstructured networks, there is no relation between the file identifier and the peer where it resides. Structured networks are more difficult to implement than an unstructured networks, mainly due to the fact that it is not easy to maintain a DHT in a highly dynamic setting. In addition to this, since data is stored typically in some arbitrary node, fault-tolerance to node deletion is essential. These are some of the reasons why structured P2P networks, despite their efficient search mechanism, have been somewhat less successful than unstructured networks when it comes to practical deployment. Hence, it is of both theoretical and practical interest to develop simple and efficient DHT schemes that work well under high churn rates.

In this paper, we present and analyze a simple and general scheme to build a churn-tolerant structured P2P network. The basic idea behind our scheme is simple. It is easy to design a static topology with desirable properties such as connectivity, low degree, low diameter, and an efficient (and local) routing algorithm. Indeed such topologies, e.g., hypercube, butterfly, cube connected cycles, de Bruijn graphs, etc., have been studied extensively in parallel and distributed computing literature. Our scheme shows how to “convert” a static graph into a fault-tolerant DHT network such that all the good properties of the static graph are guaranteed with high probability. For example, by applying our scheme to a cube-connected cycles (CCC) graph yields a $O(\log N)$ degree P2P network that has a $O(\log N)$ latency (i.e., search time), using $O(\log N)$ messages, with constant storage overhead. Here N is the expected stable network size (cf. Section 3.1). Our bounds are essentially optimal since in our model (cf. Section 3.1) if we want all nodes to access all data items with high probability, then it is necessary that the degree be $\Omega(\log N)$; otherwise there will be a non-negligible probability that there would be nodes disconnected from the system. In a dynamic network, there is the additional challenge of quantifying the work done by the algorithm to *maintain* the desired properties. An important advantage of the above protocol is that it takes $O(\log N)$ overhead (messages and time) per insertion and *no* overhead for deletions. This is optimal since, Liben-Nowell et al. [14] show that $\Omega(\log N)$ work is required to maintain (even) connectivity in this stochastic model.

Our scheme is an improvement in the degree size and message complexity over the network of Saia et al. [24]. The structured P2P network described by Saia *et al.* [24] has a $O(\log N)$ latency for search, using $O(\log^2 N)$ messages, and $O(\log^3 N)$ degree. Their fault-tolerant overlay is a butterfly-based expander topology. Their work guarantees that a large number of data items are available even if a large fraction of *arbitrary* peers are deleted (hence their scheme can tolerate even adversarial deletions unlike ours), under the assumption that, at any time, the number of peers deleted by an adversary must be smaller than the number of peers joining. In contrast, our scheme constructs a $O(\log N)$ latency and $O(\log N)$ degree P2P network that guarantees that *every* search succeeds with high probability (whp)¹ at any time, rather than just a large fraction, under a natural and general stochastic model — the $M/G/\infty$ model [22]. In a $M/G/\infty$ model the holding (session) times of nodes can have an *arbitrary* distribution, while arrival of nodes is assumed to be Poisson. (Real-world P2P network measurement studies [25, 29] have shown that this is a reasonable statistical model.) The construction of our overlay is also much simpler compared to [6, 24]. Our scheme also improves significantly on the Warp scheme [10]. Warp guarantees $O(\log N)$ search time, but has a degree of $O(\log^3 N)$. Our scheme has low maintenance overhead. In particular, node deletions does not incur any overhead. Multiple nodes can join and leave at the same time (in particular, up to a constant fraction of the total can leave and join at the same time) without any need to change the protocol, and hence our protocol can operate in a highly dynamic setting.

In a P2P network it is important to design distributed dynamic algorithms that maintain fundamental

¹Throughout, “whp” means “with probability at least $1 - N^{\Omega(1)}$ ”.

communication primitives such as spanning trees, spanners etc. For example, maintaining a breadth-first search tree is useful for efficient broadcasting, aggregation, and routing. Designing efficient distributed dynamic algorithms is challenging and only few results are known, see e.g., the work of [5] that gives a distributed dynamic algorithm for maintaining a spanner. It is non-trivial to efficiently maintain even some spanning tree dynamically — the trivial method would be to recompute a spanning tree (e.g., by breadth-first search [20]) every time the network changes. However, this will take $\Theta(D)$ time and $\Theta(|E|)$ messages [20]. In contrast, we show how a spanning tree of diameter $O(D)$ (where D is the diameter of the underlying graph) can be maintained by our scheme in $O(\log N)$ messages and $O(1)$ time per insertion or deletion. It is not clear how one can efficiently maintain a breadth-first spanning tree or some low-diameter spanning tree in many previous schemes e.g., Chord [3].

Other Related Work. The literature on DHT schemes is huge and we confine ourselves to those that appear relevant to our work. The idea of a general scheme for mapping a static network into a dynamic one has appeared before see e.g., [18, 1, 16]. The work of [27] uses a CCC graph (this is also the graph used to illustrate our general scheme in this paper) to build a structured P2P network. However the above papers do not present a rigorous analysis of the performance under a realistic stochastic model. Furthermore, to the best of our knowledge, none of the previous works, address the problem of efficient maintenance of spanning substructures under churn.

There has been other works on building fault-tolerant DHTs under different deletion models — adversarial deletions and stochastic deletions. For example, the work of [12] deals with adversarial churn and gives techniques to handle worst-case joins and leaves. Fiat and Saia [6] proposed a DHT network that is robust against adversarial deletions (i.e., an adversary can choose which nodes to fail). In this model some small fraction of the non-failed nodes would be denied from accessing some of the data items. While this solution is more general than our model it has some disadvantages: (1) It is not clear whether the system can guarantee its bounds when nodes leave and join dynamically; (2) the message complexity is large — $O(\log^3 N)$ and so is the network degree. Moreover their construction is very complicated which can increase the likelihood of error in implementation and decrease the possibility of practical use. In a subsequent paper Saia et al. [24] address the first problem and give a scheme with $O(\log N)$ time for search, using $O(\log^2 N)$ messages, and $O(\log^3 N)$ degree. Datar [4] gives a scheme based on the multibutterfly network that improves on the scheme of Fiat and Saia [6] under the adversarial deletion model. Naor and Weider [17] describe a simple DHT scheme that is robust under the following simple random deletion model — each node can fail independently with probability p . They show that their scheme can guarantee logarithmic degree, search time, and message complexity if p is sufficiently small. In contrast, our scheme is simpler than [17] and works under a more realistic stochastic deletion model (even a large constant fraction of nodes can get deleted in our model) and guarantees the same (essentially optimal) performance bounds. Also our scheme requires no maintenance overhead under deletions unlike the scheme of [17]. Hildrum and Kubiawicz [9] describe how to modify two popular DHTs, Pastry [23] and Tapestry [31] to tolerate random deletions. Finally, we point out that several DHT schemes (e.g., [28, 21, 11]) have been shown to be robust under the simple random deletion model mentioned above.

2 The Scheme

We will show how to build a P2P network $G = (V_G, E_G)$ of expected stable size $N = |V_G|$ (defined precisely in Section 3.1). Let $H = (V_H, E_H)$ be the static (“template”) graph that will be used to build G . (We will later show how the network can dynamically be made to adapt to a changing network size. Note that stable means that the total network size is more or less remains the same, up to constant factors.) We will use the term *node* to denote a node (peer) of G and the term *vertex* to denote a vertex of H .

Although, in principle, any graph can be taken as a template (or “backbone”) graph, for the purposes of constructing efficient P2P networks it is desirable that H has certain properties such as connectivity, regu-

larity, recursive structure, constant degree, logarithmic diameter, and a simple and efficient (local) routing scheme. Good candidates for H are hypercube network and its variants (butterfly, Beneš network and cube connected cycles), de Bruijn graph etc. Henceforth, the following assumptions will be made with respect to H :

1. H has diameter D and maximum degree Δ .
2. H has a local and efficient routing scheme \mathcal{R} that can route between any two nodes in $O(D)$ time using $O(D)$ messages, where D is the diameter of H . Specifically it will be required that H has a vertex labeling scheme that enables shortest path routing with low memory overhead (see e.g., [8] for a survey on such routing schemes). In such a routing scheme, vertex labels are assigned in such a way that every vertex v , given the destination address u , can decide locally (based solely on the address of u) the outgoing edge of v that (eventually) leads to u by using only a routing table of size at most $O(\Delta)$ entries per node. (Each entry of the routing table will specify which outgoing edge to take for a given destination u .) A well-known example of such a scheme is the *bit-fixing routing scheme* in a hypercube (and its variants) [13].

Given the above assumptions, our scheme builds a DHT-based P2P network (with expected stable size N) with the following properties:

- The degree of a node and its routing table size is bounded by $O(\Delta \log N)$ w.h.p. (cf. Theorem 3.2)
- At any time, the network is connected and has a diameter of $O(D)$ w.h.p. (cf. Theorem 3.2)
- Every search will succeed in $O(D)$ time w.h.p and will use $O(D)$ messages. (cf. Theorem 3.3)
- The time and message overheads for a node to join the network are $O(D)$ and $O(D + \Delta \log N)$ respectively w.h.p. (cf. Theorem 3.4)
- Number of nodes responsible for servicing a data item is $O(1)$.

Throughout, we will illustrate by taking H to be a *cube connected cycle (CCC)* network. Our scheme can be adapted to other similar types of graphs. The r -dimensional CCC is constructed from the r -dimensional hypercube by replacing each vertex of the hypercube with a cycle of r vertices in the CCC. The i th dimension edge incident to a vertex of the hypercube is then connected to the i th vertex of the corresponding cycle of the CCC [13]. In a CCC, the label of a vertex is represented by a pair $\langle w, i \rangle$ where i is the position of the node within its cycle and w is the label of the vertex in the hypercube that corresponds to the cycle. Two vertices $\langle w, i \rangle$ and $\langle w', i' \rangle$ are linked by an edge in the CCC if and only if either (1) $w = w'$ and $i - i' \equiv \pm 1 \pmod{r}$ or (2) $i = i'$ and w differs from w' in precisely the i th bit. Edges of the first type are called cycle edges, while edges of the second type are referred to as hypercube edges. A CCC graph of N nodes has diameter $O(\log N)$ and each node has degree 3. A CCC has an efficient routing scheme, namely the bit-fixing routing scheme [13] that can route in $O(\log N)$ time using $O(\log N)$ messages using only routing tables of size $O(\log N)$. In this scheme, to route a message between two vertices with vertex labels $\langle u, i \rangle$ and $\langle v, j \rangle$, the bits of u are successively transformed (say, from the first to the last) to match v . The message is routed between one dimension to the next using the hypercube edges, while the cycle edges are used to bring the message to the vertex of the cycle with the appropriate dimension.

A node x in G has a label called the *node-id* which corresponds to a vertex label of H . We will choose the size of H to be $S = |V_H| = \frac{N}{\alpha \log N}$, where $\alpha > 2$ is a constant (any such α will suffice). (Throughout we will assume logarithm to the base 2. We will omit floors and ceilings, assuming that quantity in question is rounded to the nearest integer.) Node-ids of vertices of H are assigned randomly by sampling from all possible vertex labels of H . Specifically, if H is a CCC, the node-id of a vertex is obtained as follows: toss a fair coin (has a equal probability of getting a 0 or 1) $r = \log(N/\alpha \log^2 N)$ times independently and obtain

a r -bit random bit string σ_r (r is the dimension of the CCC). Also sample a random number from 1 to r and call it i . Then the node-id of the vertex is $\langle \sigma_r, i \rangle$. We say that the node *covers* the vertex having the label corresponding to its node-id. There is an edge between two nodes with node-ids x and y if there is an edge in H between x and y or if $x = y$. (Thus note that vertices that share the same vertex label will form a clique.) We call a vertex in H to be *occupied* if there is a node in the network (i.e., a live peer) which covers this vertex; otherwise we call it to be a *hole*.

Joining and Leaving the Network. A node (say v) that wants to join the network chooses its node-id as explained above. We assume that N (the expected stable network size) or an estimate of N (a constant factor estimate is sufficient) is known to all joining nodes. Because of the numbering scheme, v can locally determine the node-ids of its (potential) neighbors without any global knowledge. v 's neighbors in the P2P networks are the nodes that cover the above determined node-ids. To join the network, v contacts any one of the nodes in the network (such entry points are provided by an external mechanism). v can then make use of an efficient routing scheme \mathcal{R} of H to find its neighbors (i.e., their IP addresses) and joins by connecting to them. If H to be a CCC, \mathcal{R} can be the standard *bit-fixing routing scheme* mentioned earlier.

A node can leave the network at any time; the node's data is transferred to a randomly chosen node with the same vertex label (note that all such nodes are neighbors of the leaving node). We show later that such a node will always exist w.h.p in our model.

Search (Look-up) Scheme. Searches are handled by a DHT scheme, similar to other DHT schemes such as Chord [3]. The data (or key) is hashed to a random vertex label in the same fashion as was done for choosing the node-id of a vertex. Data is inserted to a randomly chosen node having this label as its node-id. Search for this data is thus directed to some node (say u) having its node-id equal to the data's hashed value. The data will be stored in u or any one of the neighbors of u that share the same vertex-label. Since all nodes sharing a node-id are connected to each other (forming a clique), search will succeed even if only one node covering this vertex is live in the network (this node will have the data). Search is performed by invoking the bit-fixing routing scheme as illustrated below by an example. Suppose a node with node-id x wants to search for a data item hashed to a number t ($1 \leq t \leq S$). Let the route given by the bit-fixing routing scheme from x to t be $\langle x, u_1, u_2, \dots, t \rangle$. Then x will send a message to a neighbor node which covers u_1 which in turn will forward to its neighbor node which covers u_2 and so on.

3 Analysis

We analyze various network parameters – routing table size (i.e., degree), connectivity and diameter, maintenance overhead for joins, and the complexity for doing search. We first describe the stochastic model used in our analysis.

3.1 Model

In evaluating the performance of our protocol we focus on the long term behavior of the system in which nodes arrive and depart in an uncoordinated, and unpredictable fashion. We model this setting by a stochastic continuous-time process: the arrival of new nodes is modeled by Poisson distribution with rate λ , and the duration of time a node stays connected to the network is independently determined by an *arbitrary* distribution G with mean $1/\mu$. This is also called the $M/G/\infty$ model in queuing theory. (This is more realistic than the less general $M/M/\infty$ used in [19] to model P2P networks.) Measurement studies of real P2P systems [25, 26, 29] indicate that the above model approximates real-life data reasonably well, especially since the holding time distribution is arbitrary (in particular the study in [29] actually indicates that the holding times may follow Weibull or lognormal distributions).

Let G_t be the network at time t (G_0 has no vertices). We are interested in analyzing the evolution in time of the stochastic process $\mathcal{G} = (G_t)_{t \geq 0}$. Since the evolution of \mathcal{G} depends only λ/μ we can assume w.l.o.g.

that $\lambda = 1$. To demonstrate the relation between these parameters and the network size, we use $N = \lambda/\mu$ throughout the analysis. We justify this notation by showing that the number of nodes in the network rapidly converges to N which we call the *expected stable network size* (or simply, stable network size). We use the notation $G_t = (V_t, E_t)$ be the network at time t .

Missing proofs of Theorems and Lemmas can be found in the Appendix. It also mentions the Chernoff bounds that we use in the proofs.

3.2 Network Size

The following theorem characterizes the network size and is a consequence of the fact that the number of nodes at any time t is a Poisson distribution (this is true even if the holding times follow an arbitrary distribution) [22, pages 18-19]; applying the Chernoff bound for the Poisson distribution gives the high probability result.

Theorem 3.1 (Network Size) *If $\frac{t}{N} \rightarrow \infty$ then $E[|V_t|] = N$, and w.h.p. $|V_t| = N \pm o(N)$.*

The above theorem assumed that the ratio $N = \lambda/\mu$ was fixed during the interval $[0, t]$. We can derive a similar result for the case in which the ratio changes to $N' = \lambda'/\mu'$ at time τ .

Corollary 3.1 *Suppose that the ratio of between arrival and departure rates in the network changed at time τ from N to N' . Suppose that there were M nodes in the network at time τ , then if $\frac{t-\tau}{N'} \rightarrow \infty$ w.h.p. G_t has $N' + o(N')$ nodes.*

3.3 Network Degree

Theorem 3.2 [Degree] *At any time t such that $t/N \rightarrow \infty$, the degree of a node and the routing table size is bounded by $O(\Delta \log N)$ w.h.p., where Δ is the maximum degree of H . (If H is a CCC, then the degree is $O(\log N)$.)*

3.4 Fault-tolerance and Search

We show that every query succeeds w.h.p at any time (after a short initial period). We show this by first proving that every vertex is occupied w.h.p which ensures that queries that (logically) map to this vertex value can be serviced by some live node covering this vertex. This fact along with the way edges are constructed in the P2P network will show that a search will succeed w.h.p for every search.

The following theorem shows there is no hole w.h.p. Recall that we call a vertex as a hole (see Section 2) means that there is no node (i.e., a live peer) in the network that covers this vertex.

Lemma 3.1 (Occupancy of Vertices) *At any time t , such that $t/N \rightarrow \infty$, w.h.p. every vertex of H is occupied.*

Proof: When $t/N \rightarrow \infty$ nodes depart the network according to a Poisson process with rate 1. Also from theorem 3.1, the number of nodes in the network is at least $N - o(N)$, with probability at least $1 - 1/N^{\Omega(1)}$. Since, at any time t , every node has the same probability to occupy each of the S vertices of H , the probability that a vertex is not covered is at most

$$\begin{aligned} & \left(1 - \frac{1}{S}\right)^{(N-o(N))} (1 - 1/N^{\Omega(1)}) \\ & \leq \left(1 - \frac{\alpha \log N}{N}\right)^{(N-o(N))} (1 - 1/N^{\Omega(1)}) \end{aligned}$$

$$\leq e^{-\alpha \log N} < 1/N^2$$

by our choice of S ($\alpha > 2$).

Applying Boole's inequality (union bound), the probability that no vertex is unoccupied is at most $1/N$.

□

The following theorem on the success probability of a search query is a consequence of the previous theorem and the way nodes link to each other. Note that we assume that one time unit is taken for sending a message across an edge (i.e., one hop).

Theorem 3.3 (Search) *For any time t , such that $t/N \rightarrow \infty$, w.h.p. any search query will be successful. The time (number of hops) needed is $O(D)$ w.h.p., where D is the diameter of H .*

Proof: Consider a search query emanating at time t from the node with node-id x for a node covering a node-id y (the hash value of the data). This search will be successful if there is a path in G to one of the nodes covering y . In terms of the template graph H , consider the path from x to y given by the routing scheme \mathcal{R} of length $O(D)$. (If H is a CCC, then the \mathcal{R} is the bit-fixing scheme and D is $O(\log N)$.) This path goes through a sequence of vertices in H . From Lemma 3.1, it follows (via union bound) that every vertex of H is occupied w.h.p for a time interval $O(D)$ (starting from time t .) Thus during this time interval, every vertex in G is covered by some (live) node in the network. From our construction of G there is an edge between any node covering a vertex to any node covering the neighbor of the vertex. Thus, w.h.p the query will take $O(D)$ time. □

The above theorem also implies the following result on the connectivity and diameter of the network.

Corollary 3.2 (Connectivity and Diameter) *For any time t , such that $t/N \rightarrow \infty$, the network is connected and has a diameter of $O(D)$ w.h.p.*

Theorem 3.4 (Overhead of Joining) *For any time t , such that $t/N \rightarrow \infty$, the time and message overheads for a node to join the network are respectively $O(D)$ and $O(D + \Delta \log N)$ w.h.p.*

4 Dynamic Maintenance of Spanning Tree of Low Diameter

The scheme admits a simple local algorithm to dynamically maintain a spanning tree whose diameter is almost optimal, i.e., essentially the same as the underlying template graph, i.e., D . Note that the diameter of the P2P network is $O(D)$. Let G_t be the network at time t . The goal is to compute a spanning tree of G_t , denoted by $T(G_t)$ of diameter $O(D)$ efficiently.

The P2P network constructed by our scheme admits a very simple and efficient algorithm. Let $T(G_t)$ be the spanning tree of diameter $O(D)$ at some time t , such that $t/N \rightarrow \infty$. We will first describe how $T(G_t)$ is constructed at some time t and then describe how it is maintained under insertions and deletions at any time $t > t'$. (With a very small probability, one may have to construct the spanning tree from scratch at any time t , as discussed below. Thus strictly speaking the complexity bounds hold in an amortized sense.)

Let $S(\ell)$ be the set of nodes that share the same-vertex label ℓ . Construct a breadth-first tree T_H on the template graph H . Choose a (distinguished) node $u(\ell) \in S(\ell)$ — we call $u(\ell)$ the leader node of the set of nodes belonging to $S(\ell)$. The tree $T(G_t)$ is constructed as follows. Connect the leader nodes of the respective vertex labels as they are connected in the breadth-first tree T_H . Make all non-leader nodes of $S(\ell)$ the children of the leader node $u(\ell)$. Note that non-leader nodes will all be leaves in $T(G_t)$.

The tree is maintained as follows:

Insertion: Let a node v is inserted. Let it have vertex label ℓ . Then the node is added as a child of the leader node of $S(\ell)$. (Note that a leader exists w.h.p by Lemma 3.1.) The time and message complexity is $O(1)$ per insertion.

Deletion: Let a node w be deleted. There are two cases. If w is a non-leader node, it is simply removed. Note that this does not disconnect the tree as this will be a leaf node. On the other hand, let w be a leader node and let the vertex label of w be ℓ . Then w is deleted and in its place another (non-leader) node, say x , belonging to $S(\ell)$ (i.e., nodes that have the same vertex label ℓ) is elected as leader. By our tree construction, x will be a leaf child (again such a node will exist w.h.p by Lemma 3.1). Thus the rest of the tree is not affected. Also, by our construction, x will have an edge to w 's parent node and all its other children. Thus connectivity is preserved and diameter is still $O(D)$. The message complexity is $O(\log N)$ per deletion (since only so many nodes are affected) and the time complexity is $O(1)$. Note that leader election itself can be done in $O(1)$ rounds and $O(\log N)$ message complexity as the set $S(\ell)$ forms a clique.

There is a small probability that the above algorithm will fail, e.g., deleting a node, leaves the corresponding vertex unoccupied (i.e., a hole). In such a case, one has to reconstruct the tree from scratch as discussed first.

5 Handling Change in Stable Network Size

The performance of our scheme depends on the stability of the network. It is easy to see that our scheme can easily tolerate changes up to constant factors (thus, as mentioned earlier, it is enough to have an estimate of N up to some constant factor). However, bad events, such as the network size drastically getting reduced, possibly even leading to the network getting disconnected, can happen, but with minuscule probability in our model. In case such events happen (which will eventually happen with probability 1 if the system runs forever) remedial measures can be taken such as resorting to an external mechanism to connect the network again (if the network gets disconnected) or rejecting new connections (if the size exceeds very much) till the situation self-corrects itself. Our analysis can be extended to handle such situations.

We now discuss how the scheme can be modified to accommodate gradual changes in stable network size. As shown in Corollary 3.1, if the ratio between the arrival and departure rates in the network change, then this leads to a new expected stable network size. Suppose the new stable size is one-half of the original network size. How can the network adapt to this changed size? Assume that H is a hypercube (similar argument will work for CCC and other related networks). All that is required is to reduce S (the size of H) by a factor of 2. This can be done easily in a local manner. Each node will simply reduce its dimension by 1. This can be accomplished by dropping the last bit in the node-id. The hash values of data are also altered in the same way. It is easy to see that because of the recursive nature of construction of the hypercube (i.e., a hypercube of dimension r can be constructed from two hypercubes of dimension $r - 1$), reducing the dimension will require only $O(1)$ overhead per node. To illustrate, consider two nodes with node-ids $\langle x_1, \dots, x_r, 0 \rangle$ and $\langle x_1, \dots, x_r, 1 \rangle$. Dropping the last bit, will make *both* these nodes to cover the vertex with label $\langle x_1, \dots, x_r \rangle$. Data that were originally serviced by either of these will now be serviced by both of them. On the other hand, if the stable network size increases by a factor of two, then each node will increase its dimension by one, by adding one more random bit to its node-id (cf. Section 2). To illustrate, consider the set of nodes with the same node-id $\langle x_1, \dots, x_r \rangle$. Randomly adding one more bit (last bit), will make on the average half of the nodes in this set to cover the vertex with label $\langle x_1, \dots, x_r, 0 \rangle$ and the other half to cover $\langle x_1, \dots, x_r, 1 \rangle$. The data that are serviced by these nodes also get hashed to the same node-ids. It is not difficult to show that the above transformation preserves all the properties of the scheme, namely network degree, number of hops needed for search, fault-tolerance, connectivity, and diameter.

6 Simulation Results

In this section, we present a simulation of the scheme to get a better picture of how the network will react in practice. The theoretical results proved earlier are asymptotic, i.e., shows that the performance is good when $\frac{t}{N} \rightarrow \infty$. Thus it is also of interest to see performance data measured from simulations for the average case.

The Simulator is written in Java to mimic a network that runs for some time t with stable network size N . The network loops for t cycles, each adding a sequence of new peers, then removing peers who have

stayed for their predetermined length. Every so many cycles, the network is inspected to determine varying statistics, e.g. diameter, average degree.

Nodes arrive based on a Poisson distribution with rate λ . This is achieved by each cycle sampling a Poisson random variable with rate λ and adding that many nodes to the graph, serially. Each node's session length, $1/\mu$, is sampled then from an arbitrary distribution, with mean $1/\mu$. Based on real world statistics in Stutzbach, et al. [30, 29], Weibull, log-normal, and exponential distributions fit well to mirror actual peer session lengths. For most simulations, the session length was taken from random variable with Weibull distribution, with shape parameter $k = .59$ (based on [30]) and varying the scale parameter λ such that the mean of the random variable will be $1/\mu$. The Poisson variable rate λ and Weibull mean $1/\mu$ are then chosen so $\lambda/\mu = N$.

The simulation keeps track of the basic network statistics: diameter, average degree, as well as those of interest to this specific network construction: vertex coverage, i.e. the percentage of vertices in the "backbone" network that are covered by network nodes; average coverage, i.e. the average number of nodes covering a vertex in the "backbone"; and random path length, i.e. the average path length through the network over $\log(N)$ paths.

6.1 Coverage

The coverage (i.e., occupancy of the underlying template graph) of the network is important, without 100% coverage routing through the network cannot be assured to be done efficiently, and if the coverage becomes too low, the network may become disconnected. Coverage is measured as the percentage of vertices in the template graph (i.e., CCC) that are covered by a node in the network. Coverage is tied closely to the dimension of the CCC graph, in relation to the number of nodes in the network. If the dimension of the CCC graph is too large for number of nodes, the network can never reach 100% coverage, as seen in the dimension 10 network in Figure 1 (all figures are placed in the Appendix). However, it can also be seen that if the dimension fits the number of nodes, the graph will reach 100% coverage quickly. The dimension $r = \lceil \log(N/\log^2 N) \rceil$, with N being stable network size, gives 100% coverage once the network reaches stable size with every simulation.

6.2 Diameter

The diameter d of a CCC graph of dimension n can be computed by $d = 2n + \lfloor n/2 \rfloor - 2$ for $n \geq 4$ [7]. The diameter of the network was computed approximately by traversing the network with breadth-first search, and taking the diameter to be twice the height of the produced tree. This provides a reasonable estimate, within a constant factor. With networks of a large number of nodes, this becomes to inefficient in terms of time, in many simulations, tripling the run time. A faster approach is to consider the random path. In the random path, two nodes would be pulled from the network at random, and a path would be routed between them. $\log(N)$ paths are sampled, and their lengths averaged together. This allows a much faster measurement of the network diameter. As seen in Figure 2, the random path actually provides a much more accurate diameter than the breadth-first search; due to the efficient shape of the CCC graph, the BFS-diameter is greater by almost a factor of 2.

6.3 Average Degree and Coverage

The average degree of a node in the network can be seen in Figure 3 to grow with the network size, keeping within a constant factor of $\log(N)$. The sharp drops in the average degree occur when the network size is large enough to support a higher dimension CCC graph, spreading the nodes in the network over many more vertices in the backbone CCC graph.

The average number of nodes covering a vertex is heavily related to the average degree of a node. As seen in Figure 4, it follows the same pattern of growth. It is interesting to point out, from networks ranging of size 10000 to 150000, the average degree and coverage stay constant, around average degree of 100 and average coverage of 25.

6.4 Handling Changes in Network Size

The network will not stay at constant size forever and must compensate for drastic changes in network size. To accomplish this, the dimension of the network must be increased or decreased to adjust for an increase or decrease in overall network size. This should be accomplished in as decentralized process as possible, so that each node must work to keep track of the network stability.

We use the following method to detect changes in network size. Each time unit in the simulator, a node in the network runs a simulation method mimicking normal operations of a node. At regular intervals, the will look into its neighbors in the network and attempt to ascertain if the current network is stable, then take measures if it is not stable. The regular intervals were tested with success at 100 to 500 time units; any shorter and the fluctuation of network would interfere too greatly for one individual node to correctly calculate the network status.

A node determines whether it is stable by using the average degree of several nodes, and checking if it is close to the ideal stable degree, D . The degree of the nodes stays within $O(\log N)$, and based on previous network simulations of nodes in the networks up to 1000000 nodes, the stable degree will fall around 100, ± 50 , as seen in Figure 3. Each node tracks the progression of the average degree of a sampling of nodes in the network, called A . If A , begins moving away from the stable degree size, the node will then lower its dimension if A is falling or increase its dimension if A is rising. There is a buffer of ± 65 around D , so that random variations in the sample average degree will not trigger incorrect dimension change.

Each dimension change will only decrease the dimension of the node by 1. This is to prevent the network from growing or shrinking too quickly. If the nodes of the network dimensions would make large increases in dimension, the nodes would need to expand to too large a CCC, increasing the time the network is disconnected. Decreasing the dimension greatly would cause the cycles of the CCC to be shortened too much, causing difficulties in network routing.

If each node was left to change by themselves, many nodes would not get a chance to change or change too slowly and leave the network unstable or disconnected. To remedy this, once a node detects a network instability and changes dimension, it sends a message to each of its neighbors, suggesting for them to change their dimension to its new dimension. Each node monitors these messages, and once it receives enough of them (simulations have shown that around 5 is sufficient to eliminate any false positives), it will change its dimension to the suggested dimension, regardless of their own measure of the average degree. As each changes, it sends its own suggestion messages, which will then effectively propagate the change in dimension across the network.

As the network is undergoing change, nodes are still joining it, so their dimension is decided by rounding the average dimension of all nodes that share its vertex and node id. Since all nodes that share a vertex have the same degree, once change to the dimension comes to a vertex, they will all change very quickly, so the new node will either have the new correct dimension or be in a vertex that the change has not propagated to yet.

Figure 5 depicts a typical network response to a large change in network size. The network was simulated for 450000 time units, where a 50000 node network dropped to a 20000 node network. The network is forgiving in small decreases, but once the network drops too far at $t = 610$, the network corrects itself quickly, 70% of the nodes in the network switching in less than 7000 time units. Due to nodes continually being added while the network is adjusting, perfect instantaneous convergence to the new dimension is unlikely, but as the network progresses, it will continue to self-adjust and reduce the average dimension in

all of its nodes to the new correct dimension. The random path statistic in Figure 5 shows the average of a sampling of nodes route lengths when trying to reach a random assortment of nodes, mimicking requests during normal network operations. While the network contains nodes of differing degrees, it is still able to function normally, with few disconnects or routing problems.

7 Concluding Remarks

We presented a simple and general scheme for building a structured P2P network. We analyze our scheme under a realistic churn model and provably show that it gives essentially optimal bounds with respect to search time, degree, message complexity and maintenance overhead. The scheme offers algorithmic benefits to efficient distributed dynamic maintenance of spanning trees. It will be interesting to explore dynamic algorithms for other problems in this scheme. We also did a simulation based-study the understand the average performance of the scheme in networks of moderate size.

References

- [1] Ittai Abraham, Baruch Awerbuch, Yossi Azar, Yair Bartal, Dahlia Malkhi, and Elan Pavlov. A generic scheme for building overlay networks in adversarial scenarios. In *IPDPS*, 2003.
- [2] N. Alon and J. Spencer. *The Probabilistic Method*. John-Wiley, 1992.
- [3] H. Balakrishnan, F. Kaashoek, D. Karger, R. Morris, and I. Stoica. Looking Up Data in P2P Systems. *Communications of the ACM*, pages 43–48, February 2003.
- [4] M. Datar. Butterflies and p2p networks. In *Proceedings of the 10th European Symposium on Algorithms (ESA)*, 2002.
- [5] Michael Elkin. A near-optimal fully dynamic distributed algorithm for maintaining sparse spanners. In *26th ACM Symp. on Principles of Distributed Computing (PODC)*, pages 195–204, 2007.
- [6] A. Fiat and J. Saia. Censorship Resistant Peer-to-Peer Content Addressable Networks. In *Proceedings of SODA*, 2002. Journal version in *Theory of Computing*, vol. 3, 2007, 1-23.
- [7] I. Fris, I. Havel, and P. Liebl. The diameter of the cube-connected cycles. *Information Processing Letters*, 61(3):157–160, 1997.
- [8] C. Gavoille and D. Peleg. Compact and localized distributed data structures. *Distributed Computing*, 16(2-3):111–120, 2003.
- [9] K. Hildrum and J. Kubiawicz. Asymptotically efficient approaches to fault-tolerance in p2p networks. In *17th International Symposium on Distributed Computing (DISC)*, 2003.
- [10] S. Jagannathan, G. Pandurangan, and S. Srinivasan. Query protocols for highly resilient peer-to-peer networks. In *19th International Conference on Parallel and Distributed Computing Systems*, 2006.
- [11] M. Kashaek and D. Karger. Koorde: A simple degree optimal distributed hash table. In *IPTPS*, 2003.
- [12] F. Kuhn, S. Schmid, and R. Wattenhofer. Towards worst-case churn resistant peer-to-peer systems. *Distributed Computing*, 22:249–267, 2010. Conference version in *IPTPS* 2005.
- [13] F. Leighton. *Introduction to Parallel Algorithms and Architectures*. Morgan Kaufmann, 1992.
- [14] D. Liben-Nowell, H. Balakrishnan, and D. Karger. Analysis of the Evolution of Peer-to-Peer Systems. In *Proceedings of ACM Principles of Distributed Computing*, 2002.

- [15] D. Malkhi, M. Naor, and D. Ratajczak. Viceroy: A Scalable and Dynamic Emulation of the Butterfly. In *ACM Principles of Distributed Computing*, 2002.
- [16] G. Manku. Routing Networks for Distributed Hash Tables. In *Proceedings of the ACM Principles of Distributed Computing*, 2003.
- [17] M. Naor and U. Weider. A simple fault-tolerant distributed hash table. In *Proceedings of the 2nd International Workshop on Peer-to-Peer Systems (IPTPS)*, 2003.
- [18] Moni Naor and Udi Wieder. Novel architectures for p2p applications: the continuous-discrete approach. In *SPAA*, pages 50–59, 2003.
- [19] G. Pandurangan, P. Raghavan, and E. Upfal. Building Low-Diameter P2P Networks. *IEEE Journal on Selected Areas in Communications*, 21(6):995–1002, 2003 (Preliminary version in FOCS 2001).
- [20] David Peleg. *Distributed computing: a locality-sensitive approach*. SIAM, 2000.
- [21] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A Scalable Content Addressable Network. In *Proceedings of ACM SIGCOMM 2001*, 2001.
- [22] Sheldon Ross. *Applied Probability Models with Optimization Applications*. Dover Press, 1970.
- [23] A. Rowstron and P. Druschel. Pastry: Scalable, Decentralized Object Location, and Routing for Large-scale Peer-to-Peer Systems . In *Proc. of the IFIP/ACM International Conference on Distributed Systems Platforms*, pages 329–350, 2001.
- [24] J. Saia, A. Fiat, S. Gribble, A. Karlin, and S. Saroiu. Dynamically Fault-Tolerant Content Addressable Networks. In *Proceedings of the 1st International Workshop on Peer-to-Peer Systems*, March 2002.
- [25] S. Saroiu, P. Gummadi, and S. Gribble. A Measurement Study of Peer-to-Peer File Sharing Systems. In *Proceedings of Multimedia Computing and Networking 2002 (MMCN '02)*, San Jose, CA, USA, January 2002.
- [26] S. Sen and J. Wang. Analyzing Peer-to-Peer Traffic Across Large Networks. *IEEE/ACM Transactions on Networking*, 12(2):219–232, 2004.
- [27] H. Shen, C.-Z. Xu, and G. Chen. Cycloid: A constant-degree and lookup-efficient p2p overlay network. *Performance Evaluation*, 63(3):195–216, 2006.
- [28] I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan. Chord: A Scalable Peer-To-Peer Lookup Service for Internet Applications. In *Proceedings of the 2001 ACM SIGCOMM Conference*, pages 149–160, 2001.
- [29] D. Stutzbach and R. Rejaie. Understanding churn in peer-to-peer networks. In *Proceedings of the 6th ACM SIGCOMM conference on Internet Measurement*, 2006.
- [30] D. Stutzbach, R. Rejaie, N. Duffield, S. Sen, and W. Willinger. On unbiased sampling for unstructured peer-to-peer networks. In *Proceedings of the 6th ACM SIGCOMM conference on Internet Measurement*, 2006.
- [31] B. Zhao, J. Kubiatowicz, and A. Joseph. Tapestry: An Infrastructure for Fault-tolerant Wide-area Location and Routing. Technical Report UCB/CSD-01-1141, UC Berkeley, April 2001.

A Figures

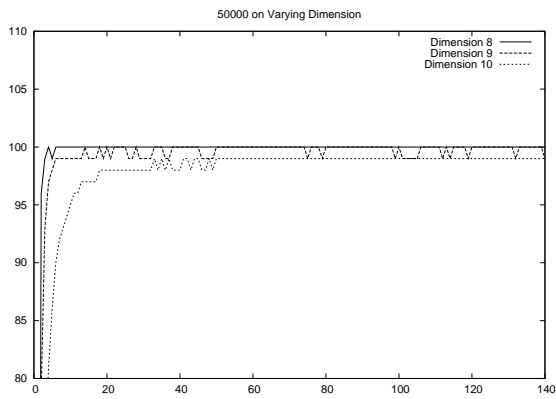


Figure 1: Coverage

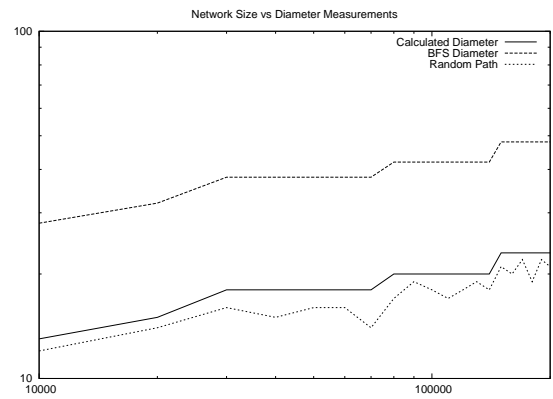


Figure 2: Diameter

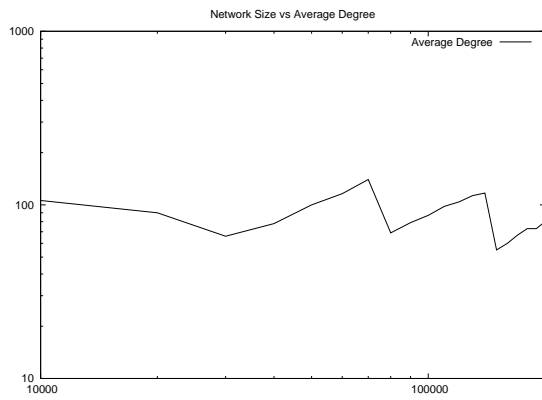


Figure 3: Average Degree

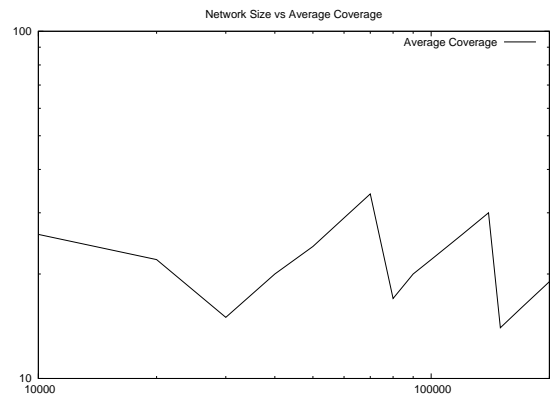


Figure 4: Average Coverage

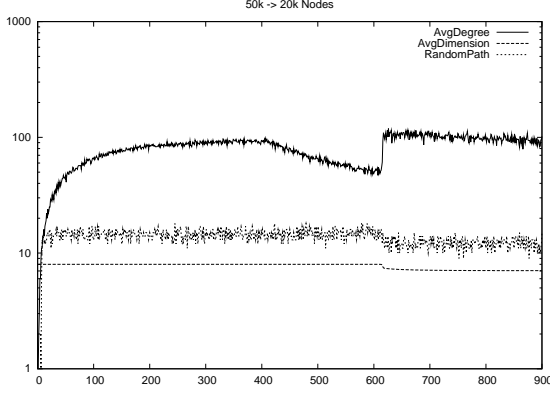


Figure 5: Dimension Adjustment

B Proofs

Throughout our analysis we use the Chernoff bounds for the binomial and the Poisson distributions. Let the random variable X denote the sum of n independent and identically distributed Bernoulli random variables each having a probability p of success. Then, X is binomially distributed with $\mu = np$. We have the following Chernoff bounds [2]: For $0 < \delta < 1$: $\Pr(X > (1 + \delta)\mu) \leq e^{-\mu\delta^2/3}$ and $\Pr(X < (1 - \delta)\mu) \leq e^{-\mu\delta^2/2}$. We have identical bounds even when X is a Poisson random variable with parameter μ [2].

B.1 Proof of Theorem 3.1

Proof: Consider a node that arrived at time $x \leq t$. The probability that the node is still in the network at time t is $1 - G(t - x)$. Let $p(t)$ be the probability that a random node that arrives during the interval $[0, t]$ is still in the network at time t , then (since in a Poisson process the arrival time of a random element is uniform in $[0, t]$),

$$p(t) = \frac{1}{t} \int_0^t (1 - G(t - x)) d\tau = \frac{1}{t} \int_0^t (1 - G(x)) d\tau.$$

Our process is similar to an infinite server Poisson queue. Thus, the number of nodes in the graph at time t has a Poisson distribution with expectation $tp(t)$ (see [22, pages 18-19]).

When $t/N \rightarrow \infty$, $E[|V_t|] = N$.

We can now use a tail bound for the Poisson distribution [2, page 239] to show that

$$\Pr\left(|V_t| - E[|V_t|] \leq \sqrt{bN \log N}\right) \geq 1 - 1/N^c$$

for some constants b and $c > 1$. \square

B.2 Proof of Theorem 3.2

Proof: We first show that the number of node-ids covering a given vertex is $\Theta(\log N)$ w.h.p. When the network is stable, the number of live nodes is at least $N - o(N)$ w.h.p, i.e., with probability at least $1 - 1/N^{\Omega(1)}$. Let Y_j be the indicator random variable for the event that some node j covers a given vertex v . Then

$$\Pr(Y_j = 1) \geq (1 - (1 - 1/S))^{N - o(N)} (1 - 1/N^{\Omega(1)})$$

where $S = \Theta(N/\log N)$, is the size of H . Thus, by linearity of expectation, the expected number of nodes covering a given vertex is

$$(N \pm o(N))(1 - (1 - 1/S))^{N-o(N)}(1 - 1/N^{\Omega(1)}) = \Theta(\log N).$$

Applying the Chernoff bound gives the high probability result.

Since the maximum degree of a vertex is Δ the number of edges incident on a node is $\Theta(\Delta \log N)$ since each node has edges to nodes that cover a constant number of vertices and each vertex covered by $\Theta(\log N)$ nodes w.h.p. The bound on the routing table size follows from the fact that H admits an routing scheme \mathcal{R} that has a routing table size of $O(\Delta)$ entries per node. \square

B.3 Proof of Theorem 3.4

Proof: An incoming node has to locate a node in the network with the same node-id; then it can find all of its neighbors in $O(1)$ time and $O(\log N)$ messages. Finding such a node (starting from some entry point node) takes $O(D)$ time (Theorem 3.3). Hence the total time needed to find all neighbors is $O(D)$. The total number of messages needed is $O(D + \Delta \log N)$ w.h.p., since D messages are needed for routing (to find a node of same id) and a routing table updates of size $O(\Delta \log N)$ has to be done in total (for the new node as well as the neighbors of the new node). \square